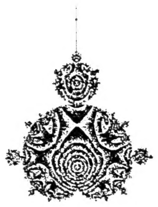

AMYGDALA $z \mapsto z^2 + c$

A Newsletter of fractals & $\mathcal{M}$ (the Mandelbrot set)
AMYGDALA, Box 219, San Cristobal, NM 87564
505/586-0197

For prices, see Amygdala PRICE LIST & ORDER FORM

Issue #17
August 15, 1989
Copyright © 1986-1989 Rollo Silver

CONTENTS

THE SLIDES (C17)

This issue's slides come from brothers Ken and Dave Philip.

547. (Ken Philip): Furry galaxy pinwheel.
586. (Ken Philip): Coil Spring. An enlarged detail from 'Phantom Wart'. This structure is the tail of a seahorse — as you descend Seahorse Valley the tails coil up more and more tightly.
Center = –0.7499385 + 0.0169434 i,
magnification = 140,000, max dwell = 7,000; dbm = 1.
Program: MandelZot version 0.95.
810. (A.G. Davis Philip): SHVR B.
Center = –0.7459,0609,1600 + 0.0984,8530,3580 i,
magnification = 8.33 x 10^9, max dwell = 3,825. Created with the new Fractal Magic program at a resolution of 640 x 480 pixels (VGA).
812. (A.G. Davis Philip): KWPELX N8; the VGA version of a connection between galaxies.
Center = –1.7699,2842,4235 + 0.0098,7853,6866 i,
magnification = 147. x 10^9, max dwell = 5,000. Program: Fractal Magic (VGA).

FRACTAL MUSIC: AN UPDATE

—*RS*

FRACTAL MUSIC, by Richard V. Robinson, appeared in Amygdala #10 (page 4). Along with the article Richard sent a tape containing a few seconds of fractal music which I found fascinating for its promise of things to come.

Last fall I received another tape from Richard, with two lovely pieces of fractal music in which the earlier promise had come to fruition.

Now Richard has sent in an article on his musical methodology, and offers his tape for sale. I strongly recommend it!

TWO METHODS FOR DERIVING MUSICAL PARAMETERS FROM THE CHAOTIC DYNAMICS OF $Z \mapsto Z^2 + C$

—*Richard V. Robinson*

Here is a writeup of the methods I used to produce the tape I sent you [RS] last October.

My first method is based on the border-following formula which created the first tape I sent you last spring, "PC Beeps of the Mandelbrot Set." I display a plot of a Mandelbrot set blow-up or Julia set. In my plots, black pixels are used exclusively to represent points inside the Mandelbrot set, or internal to a connected Julia set, those points which dwell infinitely long. Therefore there is always a well-defined border on the screen between the black and lit areas.

I select a point on this border as the starting point for my note finder, input a pace or metronome value, and seed the tune with a first note. The finder proceeds to wander the screen staying on the black points, and always keeping a lit point immediately to his left, moving four steps per metronome count.

(continued on page 10)

TURBO MANDELBROT SETS

The following article, by Dietmar Saupe, the co-author of THE SCIENCE OF FRACTAL IMAGES, is reprinted from the British publication FRACTAL REPORT by kind permission of the author.

Turbo Mandelbrot Sets[1]

Dietmar Saupe
Institut für Dynamische Systeme
Universität Bremen
2800 Bremen 33
West Germany

The Mandelbrot set (M-set for short) has recently attracted considerable attention from amateur scientists and home computer programmers, in large part due to an August 85 article in Scientific American and its associated cover picture [*Dewdney 85*]. Since then, it has become clear that the M-Set and its cousins, the Julia-sets, are capable of generating images with fascinating self-similarities and curious yet natural-looking shapes[2].

Probably more important, though, is that the complexity of these pictures, as measured by the length of the programs capable of generating them, is quite small. Consequently, since the *Scientific American* article was published, such simple do-loops as the following have surfaced:

```
#define LARGE 1000.0
int GetIterations (cx, cy, maxiter)
double cx, cy;
int maxiter;
{
    int iter = 0;
    double x, y, x2, y2;
    x = y = x2 = y2 = 0.0;
    while ((x2 + y2 < LARGE) && (iter < maxiter)) {
        y = 2 * x * y + cy;
        x = x2 - y2 + cx;
        x2 = x * x;
        y2 = y * y;
        iter++;
    }
    return (iter);
}
```

These do-loops have surely devoured thousands of CPU hours on computers of all sizes. Also, some public domain M-set software packages have appeared since that time. With these, people have been able to compute, store, and retrieve images – but often only a substantial cost. For instance, I recently tried one such program (MandelColor 3.0 from Robert Woodhead Inc.), selecting a small section of the M-Set before starting the computation. First, the screen was cleared, and then the following message appeared:

Now computing the Mandlebrot (sic!) set.
This will take a while for large windows.
Please be (very!) patient.

And, indeed, the picture was generated pixel-by-pixel, line-by-line. The message was definitely to be taken seriously.

Here we describe a new approach to faster M-Set computation, based on the exposition of Y. Fisher in *The Science of Fractal Images [Fisher 88].* A beefed-up Macintosh II version of this same approach, containing several algorithms in addition to a color palette editor, is offered in *The Game of Fractal Images [Parmet 88].*

The two images on the following page offer an example. The Mandelbrot set in the first blow-up (Figure 1) was enlarged by a factor of 760,000. The lower left corner of the image is at (-1.773 520 992, 0.006 835 510), the width is

[1] This paper is an extended and modified reprint of an article which appeared in the Silicon Graphics Iris Universe, Fall 1988.

[2] For a detailed account of Julia sets and related topics with many full color pictures see *The Beauty of Fractals* by H.-O. Peitgen and P. Richter, Springer-Verlag, New York 1986.

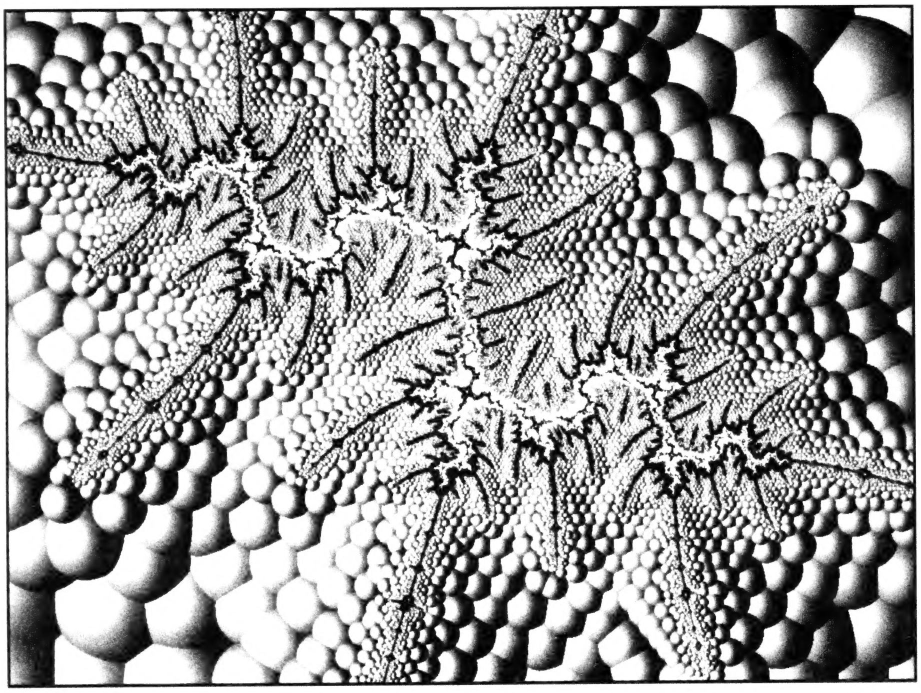

Figure 1: Blow up of the Mandelbrot set, magnification factor is approximately 760,000. See text for details.

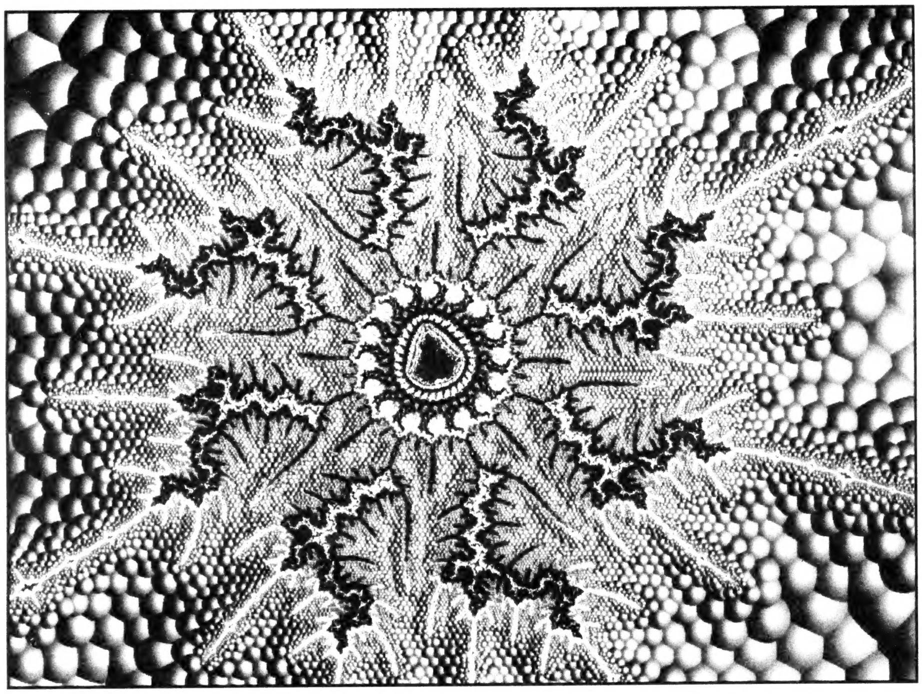

Figure 2: Blow up of the Mandelbrot set (small detail of Figure 1), magnification factor is approximately 1,700,000,000. See text for explanations.

0.000 003 931. Thanks to the new M-Set computation approach, only about 13 minutes of total CPU time was required on an IRIS 3130[3] to produce the image, plus three to four minutes to render the spheres (each point was computed

[3]The IRIS 3130 is a graphics workstation from Silicon Graphics, Inc. based on the 68020 Motorola processor, a custom VLSI chip for fast floating point calculations and special hardware, called *geometry engines* for support of 2D- and 3D-graphics operations (these are not used in our application discussed here).

Entries for the event queue:	
LEFTMOUSE	place the pixel at the mouse position on the highest priority stack; the Turbo algorithm will process that pixel next, drawing a disk.
DKEY	toggle between single and double-precision calculation.
AKEY	toggle between Turbo method and usual iteration without disks.
ZKEY	output number of iterations accumulated, compare against the number that would be necessary with the standard algorithm (can be computed from the stored array of pixel values).
MIDDLEMOUSE	invokes pop-up menu.
Entries for the pop-up menu:	
Continue	proceed with the computation of the image (otherwise the pop-up menu is brought up again to allow more interaction).
Maxiter	brings up another pop-up menu of numerical values to be chosen for the number maxiter (the maximum number of iterations allowed per pixel).
Recur	brings up a menu for selection of the parameter recur (determines the smallest disks allowed to define subsequent candidate points around its boundary).
Zoom	lets the user define a section of the current image to be used as the next world-window in the algorithm; the present computation should be aborted.
Save	load an image and the world-window coordinates in one or two files.
Load	load an image which previously had been stored on disk, the coordinates are also retrieved such that further zooms are possible.
Show Disks	enters a mode in which disks at the current mouse position are computed and drawn in a highlighted color.
Clear	clears the screen and memory to start a fresh picture (maybe with different parameter settings).
Job	collect all information (window, parameters, etc.) about the current image and store it in a file, which later can be processed by a batch version of the program.
Quit	exit the program.

Table 1: Suggested table of user interactions. The user interacts with the program by means of mouse and keyboard buttons in addition to pop-up menus. LEFTMOUSE and RIGHTMOUSE refer to the left and right button of a 3-button-mouse. The event queue records state changes of all listed buttons, followed by the indicated actions.

separately). The image in Figure 2 is a blow-up of the pixel taken from the center of Figure 1, enlarged by a total factor of 1,700,000,000. The lower left corner of the image is at (-1.773 519 041 423, 0.006 836 860 241), where the width is 0.000 000 001 765. Total CPU time on the 3130 came to about 43 minutes.

For a fast algorithm, it would be extremely valuable to be able to compute the distance $d(c, M)$ of a point (pixel) c from the M-Set M, like so:

$$d(c, M) = \inf \{ \sqrt{(x - \mathrm{Re}\, c)^2 + (y - \mathrm{Im}\, c)^2} | x + iy \in M \} .$$

Knowing the distance, then, we could exclude a disk entered at the pixel c with the radius $d(c, M)$ from further computations, since none of the pixels inside the disk belongs to the M-Set. Given that the disk may contain dozens or even thousands of pixels, you can imagine the potential savings such an exclusion represents. Also, although no formula exists for calculating the distance, there is an estimate for it, namely:

$$R = \frac{\sinh G(c)}{2 \exp(G(c)) |G'(c)|} < d(c, M) .$$

```
ALGORITHM    TurboMSet ()
Title        Fast computation of Mandelbrot set
Globals      recur          parameter of the algorithm (see MDisk)
             xmin, xlen     start and length of window in x
             ymin, ylen     start and length of window in y
             data           pointer to doubly indexed array of shorts
             ixlen, iylen   integer dimension of viewport
             pixel          real, size of a pixel
             maxiter        maximal number of iterations allowed
Variables    ix, iy         integers
Functions    push (ix, iy, size) checks if (ix,iy) is not already marked
             in the data array and (if not) puts the coordinates (ix,iy)
             on one of the stacks (large values of size will select higher
             priority stacks)
             pop (ix, iy) pops the top entry of the high priority stack
BEGIN PROGRAM
   ... initialize the graphics routines
   ... define the world window coordinates
   ... allocate memory for the data array (size ixlen by iylen)
   ... initialize stacks of candidate pixels
   pixel := xlen / (ixlen - 1)      /* size of pixel */
   push (0, iylen-1, 100)           /* upper left corner on stack */
   push (ixlen-1, 0, 100)           /* lower right corner on stack */
   push (ixlen-1, iylen-1, 100)     /* upper right corner on stack */
   FOR iy=0 TO iylen-1 DO
      FOR ix=0 TO ixlen-1 DO
         IF data[ix][iy] = 0 THEN
            MDisk (ix, iy)
            IF event queue not empty THEN
               ... process user interaction
            END IF
            WHILE queue empty and stacks not empty DO
               pop (ix, iy)    /* pop stack */
               IF data[ix][iy] = 0 THEN
                  MDisk (ix, iy)
               END IF
            END WHILE
         END IF
      END FOR
   END FOR
END PROGRAM
```

Therefore, a disk drawn around c with radius R also does not intersect the Mandelbrot set. In this formula, $G(c)$ denotes the potential of M at the point c, the negative base 2 logarithm of which is essentially the number of iterations computed by **GetIterations** () above:

$$G(c) = \lim_{k\to\infty} \frac{\log|z_k|}{2^k},$$

where

$$z_{k+1} = z_k^2 + c, z_0 = c.$$

In computing R, further simplifications are possible. The result is a scheme about twice as costly as the usual iteration above, plus the cost for one evaluation of a logarithm and a square root (see *The Science of Fractal Images* for details and pseudo-code).

Given a routine to compute R, one can use the following "Turbo-fast" scheme to create M-Sets: start with the first pixel in first scan line, compute radius R, and draw a disk of that radius. Then, proceed to the next pixel on the first scanline not covered by the disk. Again, compute and draw a disk. Continue in this fashion for the remaining pixels in the first and, later, in all other scanlines.

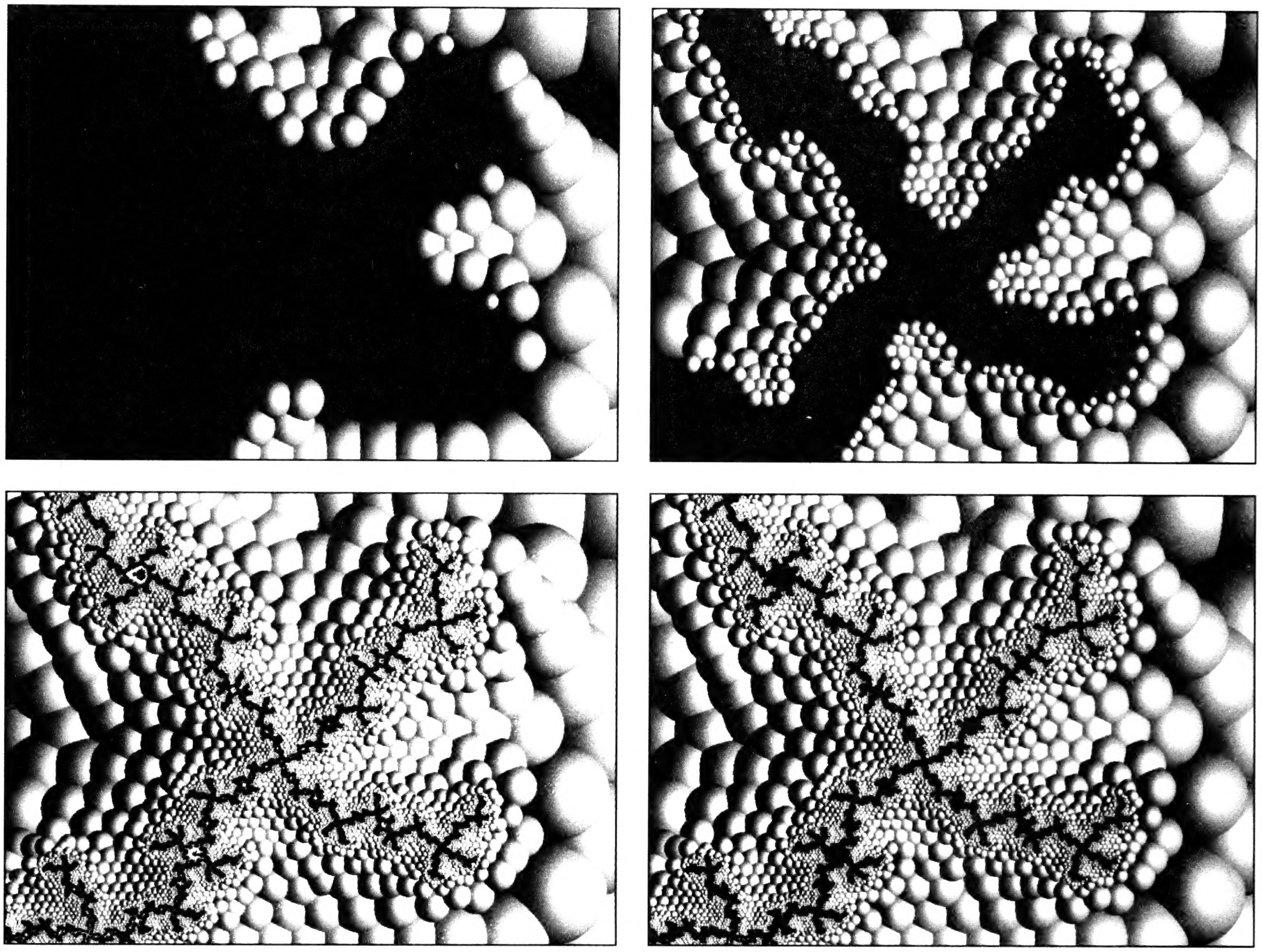

Figure 3: Here is a clockwise sequence illustrating the progression of the Turbo algorithm in four snapshots. See end of text for details.

A more interesting approach results in the following recursive method: for each disk drawn select some pixels next to the disk (we have used six pixels in the images presented here). For each point (not already covered by a disk), recursively draw disks and determine more candidate points along their boundaries. Recursion should stop when the disks become smaller than some predefined tolerance, a parameter of the algorithm (delta). When no more candidate pixels are to be processed, the next step is to scan through the image to see which pixels have not yet been covered. For those, recursion can be resumed until the picture is complete (see the pseudo-code **TurboMSet()** and **MDisk()** for more details). A few more points may also prove useful:

1. The color of the disks used in the approach just described should be uniform; for example, you can use the number of iterations as an index for a color lookup table.

2. While the computation is proceeding, you can input seed pixels by clicking a mouse button, thus instructing the algorithm to go to those locations first.

3. It is somewhat better to work with several stacks as compared to using pure recursion. Depending on the size of the currently drawn disk, you may enter candidate pixels around the boundary into one of the stacks. The algorithm always checks first for those candidates that originate from the largest disks. That way larger disks are drawn first, and a rough global picture can be quickly obtained.

```
ALGORITHM    MDisk (ix, iy)
Title        Compute and draw a disk around given pixel
Arguments    ix, iy         integer coordinates of pixel
Globals      as above in TurboMSet
Variables    R              estimated radius of disk
             rad            radius in pixels (integer)
             idx, idy       integers
             iter           number of iterations
             cx, cy         real numbers, pixel coordinates
             ColorIndex     integer index for color lookup table
             r              integer
Functions    MSetRad (cx, cy, maxiter, R, iter) computes radius R of disk
             around (cx,cy) completely outside of M-Set and number of
             iterations iter (see [3])
             FillDisk (ix, iy, rad, index) fills a disk centered at
             (ix,iy) with radius rad on the screen with color given by
             index, fills corresponding disk in data array data[][]
             DrawPoint (ix, iy, index) same as FillDisk for individual
             pixel (ix,iy)
             push (ix, iy, size) see TurboMSet
BEGIN PROGRAM
    cx := xmin + ix * pixel            /* world x coordinate of point */
    cy := ymin + iy * pixel            /* world y coordinate of point */
    MSetRad (cx, cy, maxiter, R, iter) /* compute R, iter */
    ColorIndex := iter
    rad := (int) (R / pixel)           /* radius in integer pixel units */
    IF rad > 0 THEN
        FillDisk (ix, iy, rad, ColorIndex)
    ELSE
        DrawPoint (ix, iy, ColorIndex)
    END IF
    /* if the radius is still bigger than recur pixels, then push 6
       new pixels around the boundary of disk on the stacks */
    IF R > recur*pixel THEN
        r := rad + 1
        idx := (int) 0.500 + 0.500 * r
        idy := (int) 0.500 + 0.866 * r
        push (ix + r, iy, rad)
        push (ix - idx, iy + idy, rad)
        push (ix - idx, iy - idy, rad)
        push (ix + idx, iy + idy, rad)
        push (ix - r, iy, rad)
        push (ix + idx, iy - idy, rad)
    END IF
END PROGRAM
```

4. Enter pixels into one of the stacks only if the pixel is in the window and not already tested.

5. The algorithm should always be prepared to accept user input for defining the subsection that is to be computed next.

6. At some point (when all further disks seem to be too small to care about), you may want to disable the computation of disk radii in order to achieve a speed-up of nearly two-fold. This can be done automatically when all stacks are empty. From then on, however, one has to check periodically for possible disks.

7. There are two ways to check a pixel to see if it already is set: either disks have been drawn within an internal array in CPU memory or they have been drawn only in the image memory (frame buffer). Typically, as on an IRIS 3000 system, they should been drawn the first way because it is faster to read CPU memory as opposed to the

frame buffer (for example, the Bresenham circle drawing algorithm, as detailed in *Computer Graphics [Hearn 86]*, is a good option). But whenever disks have been drawn in CPU memory, notice that the result in the image will not be identical to the outcome of the IRIS **circfi**() routine. Thus, to draw the picture one should rather copy data from the internal array into the frame buffer.

For aesthetic reasons all the images presented in this article were rendered using shaded and Z-buffered spheres in place of disks. The projection of the spheres onto the plane $z = 0$ corresponds exactly to the disks described above. Finally, there is also a (more complicated) estimation formula that can be used to calculate for the distance between a point inside the M-Set and the M-Set's boundary. Thus, besides filling the outside of the M-Set with disks, one can also quickly fill the interior. Notice also that very similar ideas can be applied to the computation of connected Julia sets (see *[Fisher 88]* and *[Parmet 88]*).

Figure 3 shows a clockwise sequence illustrating the progression of the Turbo algorithm in four shots. The lower left corner of the upper left image is at (0.350 319, 0.658 221), the width is 0.043 271, maxiter = 200, recur = 1, and resolution = 768 by 576. The number of iterations used in the Turbo algorithm, moving clockwise from the upper left image are: (in the upper left image) 950, (in the upper right image) 5400, and (in the lower right image) 280,000. In the lower left image, another 870,000 standard iterations were used to finish the picture shown on the lower right. Thus the overall cost is roughly 2 * 280 K + 870 K = 1430 K standard iterations. This compares to 7000 K iterations used by the standard algorithm. The improvement, by a factor of approximately five, may not seem all that great at first blush, but one has to consider that most of CPU time under the Turbo approach is spent filling in very fine detail, whereas the overall shape is generated quite quickly. In interactive experimentation, therefore, one should never have to wait for the conclusion of the program. The total time on an IRIS 3130 to produce the final 1,150,000-iteration image shown here is about 70 seconds (using disks, not spheres).

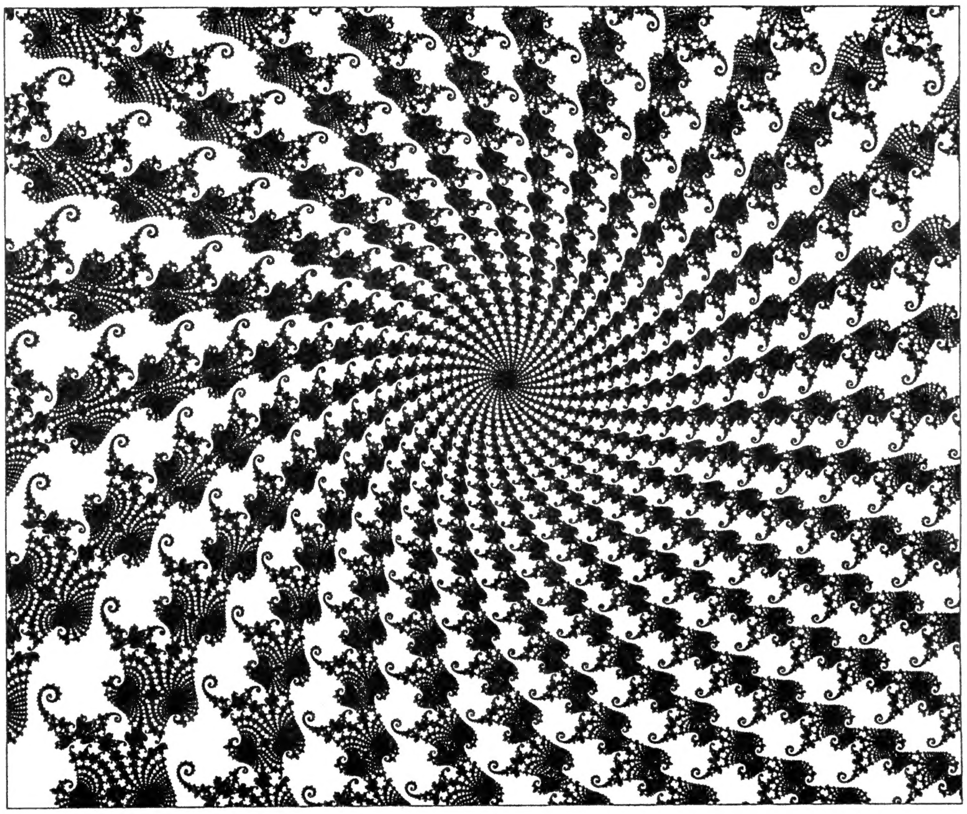
Figure 4: Example of a result of the simplified algorithm from the end of the paper. This section is centered at (-0.74691, 0.10725) and the width is 0.00134.

Finally, let us briefly mention a version of the program which computes only the boundary of the Mandelbrot set as shown, for example, in Figure 4. It is even faster and suitable for PCs and other small computers. The method relies on the fact, that the Mandelbrot set is connected. Starting out from given pixels in the M-Set or its boundary it checks out neighboring pixels, then neighbors of the neighbors and so forth. In this way computations away from the Mandelbrot set are almost totally avoided resulting in another considerable speedup. However, only a black and white image is obtained. As before, a stack of candidate pixels is maintained. When a pixel is popped from the stack, it has to be processed in the following five steps.

1. Record in an array (can be a bitmap) that the pixel is being tested, i. e. turn on the bit representing the pixel.
2. Compute the radius R of a disk around the pixel. This disk must not intersect the M-Set (i. e. one call to **MSetRad**() is executed).
3. If $R > delta$ ($delta$ is the parameter of the algorithm which determines the thickness of the drawn boundary of the M-Set), then no action is taken and the next candidate pixel can be popped from the stack. Otherwise, go to step 4.
4. The pixel belongs to the representation of the M-Set or its boundary and is colored accordingly.
5. All neighboring pixels, which belong to the image and which have not been tested before are added to the stack of candidate pixels.

This algorithm uses two bitmaps, one keeps track of the tested pixels, the other eventually contains the image of the M-Set and its thickened boundary. Initially, both bitmaps must be zeroed. To initialize the stack of candidate pixels for a blow up of the Mandelbrot set one has to trace the whole boundary of the image for pixels with $R \leq delta$. These are entered into the stack and the bitmaps. For this initialization one can exploit the fast scheme explained in the main part of this article.

References

[Dewdney 88] A.K. Dewdney, "Computer Recreations: Exploring the Mandelbrot Set", *Scientific American* (August 1985).
[Fisher 88] Y. Fisher, "Exploring the Mandelbrot Set", *The Science of Fractal Images*, H. O. Peitgen, D. Saupe (editors), pp. 287 - 296, Springer-Verlag, New York (1988).
[Hearn 86] D. Hearn, M. P. Baker, *Computer Graphics*, Prentice-Hall, Englewood Cliffs (1986).
[Parmet 88] M. Parmet, H. O. Peitgen, H. Jürgens, D. Saupe, "The Game of Fractal Images", available on computer disk through Springer-Verlag, New York (1988).
[Peitgen 86] H. O. Peitgen, P. Richter, "The Beauty of Fractals", Springer-Verlag, Heidelberg (1986).

For each step the finder takes, an event occurs. This event's nature depends on the relationship between the last point visited and the current one, according to these rules:

- ☛ If the current step constituted a counter-clockwise movement relative to the origin, the event is a silence, otherwise, the event is a sounding interval.

- ☛ For sounds, if the current step moved the finder away from the origin, the interval will be a downward one, making a note lower than the one last sounded. If he moved toward the origin, the interval moves upward.

- ☛ Intervals are selected from an array of 15 equal-tempered possibilities, not necessarily all different, using the color value of the nearest lit point as an index.

My other method involves tracking the trajectories of several different points as they are iterated through the formula. I begin with a group of points arranged in a circle anywhere on the plane. Using each of these points as a c and initial z, I iterate $z \mapsto z^2 + c$ until all points have escaped or cycled, or until memory runs out, saving the points of the trajectories as well as keeping track of the maximum and minimum modulus (distance from the origin) for each trajectory. Each trajectory is assigned to one of four functions for a voice: Sound/Silence, Pitch, Duration, or Volume. I sort them so that longer trajectories (usually those which never escaped or cycled) are used for pitch, and shorter ones for Sound/Silence. The range of modulus values for each trajectory is mapped into a array of values appropriate to the trajectory's voice function, e.g. $\{0, 1\}$ for Sound/Silence or $\{G, A, B, C, D, E, F\#, G\}$ for Pitch.

Finally, the trajectories are traversed from their beginnings, evaluating their voice functions. Each iteration produces an event of silence or sound for some duration at some pitch and volume. At the ends of the shorter trajectories, I loop back to their beginnings. At the end of the longest trajectory, the piece ends.

I should note that I program and select synthesizer voices by hand. My synths are a Yamaha TX81Z and a Kawaii Kim amplified through my home stereo system. The voices I've used on the tape were selected mainly for their dramatic value, my intention being to make something that would be listenable at least one time through.

I feel that my first method generates fairly melodic output, with recognizable musical elements such as rhythmic variety and repetition, thematic transposition, elaboration, inversion, and retrograde motion. Its most serious flaw is its strictly monophonic output. I've attempted to ameliorate this by having it sustain some notes.

My second method represents a first attempt at generating polyphony. It doesn't really do this very musically, since the multiple voices produced are completely unrelated. I have found that assigning percussion sounds to all or all but one of the trajectories produces moderately listenable percussion music.

As ever, I am interested in any work being done in this area by your correspondents. I'm continuing my work on making more musical fractal music, and will certainly provide you with news of whatever I come up with. I can provide copies of the tape to anyone interested for $7.00. Please refer inquiries to me at my business address:

Richard V. Robinson / P.O. Box 31700 / Seattle, WA 98103

REVIEW OF *ADVANCES IN COMPUTER GRAPHICS III*

—*Ian D. Entwistle*

Advances in Computer Graphics III, M.M. de Ruiter (Ed.), *Springer Verlag* (1988).

Chapter (or Part) 5, pp. 177-205, is an excellent tutorial on fractals. Entitled *Fractals — Mathematics, Programming and Applications*, it covers three topics. The first topic is the mathematics of fractals. Several classes, e.g. Koch islands, complex mappings (Mandelbrot and Julia) and strange attractors are discussed. The generation and visualization of fractal objects is the second subject. The interactive generation and representation of fractal objects is described. The third topic is the application of fractals to modelling objects from nature. In this part some examples are given of how fractals may serve as mathematical models for biological objects.

The frontispiece of the book is a lovely color photo of $a \sin z + b$.

This book is a great find.

PRODUCTS

ART MATRIX; PO Box 880 / Ithaca, NY 14851 / USA. (607) 277-0959. "Nothing But Zooms" video, Prints, FORTRAN program listings, postcard sets, slides. Send for FREE information pack with sample postcard. Custom programming and photography by request. Make a bid.

CIRCULATION

As of August 15, 1989, Amygdala has 750 paid-up subscribers, 145 of whom have the supplemental color slide subscription.